Apple Assembly Line

$1.80

Volume 5 -- Issue 6 March, 1985

In This Issue...

Videx Ultraterm Driver

We've just completed a Videx Ultraterm display driver for S-C Macro Assembler Version 2.0, so now you fine-print fans can use the assembler with that card's high-density modes. (My favorite is the 48 x 80 inverse mode.) As with the other Version 2.0 drivers, complete source code is supplied so you can tailor the card's performance to your tastes.

This driver is included on all Version 2.0 disks after number 1274. Those of you with lower serial numbers can return your original disk for updating. Please include $1.00 to cover postage and handling. We have also corrected several minor assembler bugs in the last month, so those of you with serial numbers below 1252 might want to update your disks as well.

Quarterly Disk #18

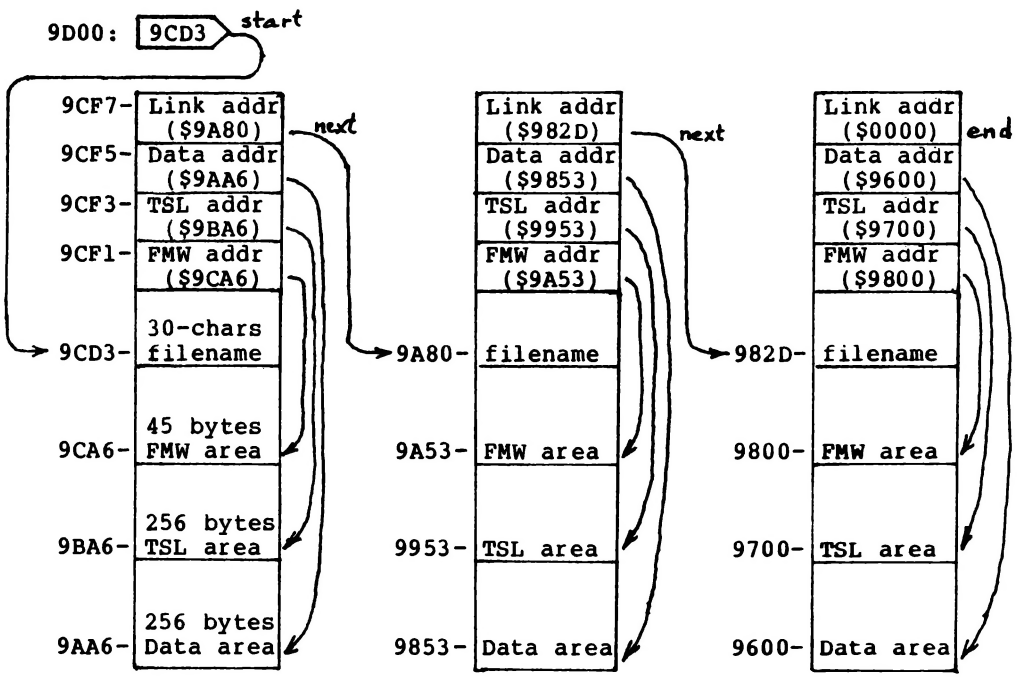
I'd also like to remind you that AAL Quarterly Disk #18 is now ready. This disk contains all of the source code from the January through March '85 issues, including the final installments of DP18 and this month's 65816 disassembler. That's many hours' worth of typing saved, at a cost of only $15. Remember that we also sell a year's subscription to the Quarterly disks for only $45. That's four disks for the price of three!

Shortening the DOS File Buffer Builder.....Bob Sander-Cederlof

Lately I have been looking through DOS for subroutines that can be shrunk. There seem to be a lot of them, or at least I have been lucky in finding some easy ones with little trouble. Elsewhere this month I show how to shrink the numeric input conversion routine, saving enough bytes to make room for a useful new feature.

Yesterday I happened across the file buffer initializer, which starts at $A7D4 and goes up to $A850. Scanning quickly through the code it looked a likely candidate for the shrinking process. If you take a quick peek, you'll see that it starts out with an SEC instruction that is totally unnecessary. Already we have shaved off one byte!

The DOS file buffers are each 595 bytes, linked together with a chain of pointers. There are normally three buffers, starting at $9600, $9853, and $9AA6. (If you have "Beneath Apple DOS", look on page 6-13 for some explanation.) Each buffer contains a 256 byte area for data, another 256 byte area for a track/sector list, a 30-character filename, a 45-byte working area for the DOS File Manager, and 4 2-byte pointers. There is a two-byte pointer kept at $9D00,9D01 which points at the first character of the filename in the highest buffer. This is normally $9CD3. Here is a picture of the normal three buffers, all chained together:



S-C Macro Assembler Version 2.0.....$100
Version 2.0 Upgrade Kit for 1.0/1.1/1.2 owners.....$20
Source Code for Version 1.1 (on two disk sides).....$100
Full Screen Editor for S-C Macro (with complete source code).....$49
S-C Cross Reference Utility (without source code).....$20
S-C Cross Reference Utility (with complete source code).....$50
DISASM Dis-Assembler (RAK-Ware).....$30
Source Code for DISASM.....additional $30
S-C Word Processor (with complete source code).....$50
DP18 Source and Object.....$50
Double Precision Floating Point for Applesoft (with source code).....$50
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk.....$15

(All source code is formatted for S-C Macro Assembler. Other assemblers require some effort to convert file type and edit directives.)

AAL Quarterly Disks.....each $15, or any four for $45

		Jan-Mar	Apr-Jun	Jul-Sep	Oct-Dec
Each disk contains	1980	-	-	-	1
the source code from	1981	2	3	4	5
three issues of AAL,	1982	6	7	8	9
saving you lots of	1983	10	11	12	13
typing and testing.	1984	14	15	16	17
	1985	18			

AWIIe Toolkit (Don Lancaster, Synergetics).....$39
Visible Computer: 6502 (Software Masters).....(reg. $50) $45
ES-CAPE: Extended S-C Applesoft Program Editor (new price, was $60) $40
"Bag of Tricks", Worth & Lechner, with diskette.....($39.95) $36

Blank Diskettes (Verbatim)..... package of 20 for $32
(Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each.....10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).....40 cents each
(Cardboard folders designed to fit 6"X9" Envelopes.) or $25 per 100
Envelopes for Diskette Mailers..... 6 cents each

quikLoader EPROM System (SCRG).....($179) $170
PROmGRAMER (SCRG).....($149.50) $140
D MAnual Controller (SCRG).....($90) $85
Switch-a-Slot (SCRG).....($190) $175
Extend-a-Slot (SCRG).....($35) $32
Write Guard Disk Mod Kit (Mark IV)..... $40

Books, Books, Books.....compare our discount prices!

"Inside the Apple //e", Little.....($19.95) $18
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner.($19.95) $18
"Apple][Circuit Description", Gayler.....($22.95) $21
"Understanding the Apple II", Sather.....($22.95) $21
"Enhancing Your Apple II, vol. 1", Lancaster.....($15.95) $15
Second edition, with //e information.
"Assembly Cookbook for the Apple II/IIe", Lancaster.....($21.95) $20
"Incredible Secret Money Machine", Lancaster.....($7.95) $7
"Beneath Apple DOS", Worth & Lechner.....($19.95) $18
"Beneath Apple ProDOS", Worth & Lechner.....($19.95) $18
"What's Where in the Apple", Second Edition.....($19.95) $19
"6502 Assembly Language Programming", Leventhal.....($18.95) $18
"6502 Subroutines", Leventhal.....($18.95) $18
"Real Time Programming -- Neglected Topics", Foster.....($9.95) $9
"Microcomputer Graphics", Myers.....($12.95) $12
"Assem. Lang. for Applesoft Programmers", Finley & Myers($16.95) $16

Add $1.50 per book for US shipping. Foreign orders add postage needed.

Texas residents please add 6 1/8 % sales tax to all orders.

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
*** (214) 324-2050 ***
*** We accept Master Card, VISA and American Express ***

The file buffer initializer gets called during the boot procedure, and by the MAXFILES command processor. There are two input parameters: the start of buffers address at $9D00, and the number of file buffers at $AA57. The job of the initializer is to fill in the four address values at the top of each buffer, to store a 00 byte in the first character of the filename of each buffer, and to store a new value in the HIMEM variable for the current language. Here's the way it was, without comments.

```
1000 *SAVE S.INIT BUFFERS
1010 *---------------------------
1020 PNTR         .EQ $40,41
1030 HIMEM        .EQ $4C,4D
1040 FP.STRINGS   .EQ $6F,70
1050 FP.HIMEM     .EQ $73,74
1060 PP           .EQ $CA,CB
1070 *---------------------------
1080 BUF.START           .EQ $9D00
1090 NO.FILES            .EQ $AA57
1100 TEMP                .EQ $AA63
1110 ACTIVE.BASIC.FLAG   .EQ $AAB6
1120 *---------------------------
1130         .OR $A7D4
1140         .TA $08D4
1150 *---------------------------
1160 INIT.FILE.BUFFERS
1170         SEC
1180         LDA BUF.START
1190         STA PNTR
1200         LDA BUF.START+1
1210         STA PNTR+1
1220         LDA NO.FILES
1230         STA TEMP
1240 *---------------------------
1250 .1      LDY #0
1260         TYA
1270         STA (PNTR),Y
1280         LDY #$1E
1290         SEC
1300         LDA PNTR
1310         SBC #$2D
1320         STA (PNTR),Y
1330         PHA
1340         LDA PNTR+1
1350         SBC #0
1360         INY
1370         STA (PNTR),Y
1380         TAX
1390         DEX
1400         PLA
1410         PHA
1420         INY
1430         STA (PNTR),Y
1440         TXA
1450         INY
1460         STA (PNTR),Y
1470         TAX
1480         DEX
1490         PLA
1500         PHA
1510         INY
1520         STA (PNTR),Y
1530         INY
1540         TXA
1550         STA (PNTR),Y
1560         DEC TEMP
1570         BEQ .2
1580         TAX
1590         PLA
1600         SEC
1610         SBC #$26
1620         INY
1630         STA (PNTR),Y
1640         PHA
1650         TXA
1660         SBC #0
1670         INY
1680         STA (PNTR),Y
1690         STA PNTR+1
1700         PLA
1710         STA PNTR
1720         JMP .1
1730 *---------------------------
1740 .2      PHA
1750         LDA #0
1760         INY
1770         STA (PNTR),Y
1780         INY
1790         STA (PNTR),Y
1800         LDA ACTIVE.BASIC.FLAG
1810         BEQ .3
1820         PLA
1830         STA FP.HIMEM+1
1840         STA FP.STRINGS+1
1850         PLA
1860         STA FP.HIMEM
1870         STA FP.STRINGS
1880         RTS
1890 *---------------------------
1900 .3      PLA
1910         STA HIMEM+1
1920         STA PP+1
1930         PLA
1940         STA HIMEM
1950         STA PP
1960         RTS
1970 *---------------------------
```

I rearranged the code, kept mental track of carry status, optimized register usage, and lopped off 11 bytes. Speed is no issue, because it is not a time critical operation anyway, but mine may be a tad quicker. Compare the two versions, and you can learn a few tricks for your own use.

```
                    1000 *SAVE S.INIT BUFFERS (S-C)
                    1010 *-------------------------------
                    1020 *   REPLACEMENT FOR DOS 3.3 CODE
                    1030 *      (SAVES 11 BYTES, NO CHANGE IN FUNCTION)
                    1040 *-------------------------------
40-                 1050 PNTR              .EQ $40,41
4C-                 1060 HIMEM             .EQ $4C,4D
6F-                 1070 FP.STRINGS        .EQ $6F,70
73-                 1080 FP.HIMEM          .EQ $73,74
CA-                 1090 PP                .EQ $CA,CB
                    1100 *-------------------------------
9D00-               1110 BUF.START         .EQ $9D00
AA57-               1120 NO.FILES          .EQ $AA57
AA63-               1130 TEMP              .EQ $AA63
AAB6-               1140 ACTIVE.BASIC.FLAG .EQ $AAB6
                    1150 *-------------------------------
                    1160         .OR $A7D4
                    1170         .TA $08D4
                    1180 *-------------------------------
                    1190 INIT.FILE.BUFFERS
A7D4- AD 57 AA      1200         LDA NO.FILES      DO (NO.FILES) TIMES
A7D7- 8D 63 AA      1210         STA TEMP          USE TEMP FOR COUNTER
A7DA- AD 00 9D      1220         LDA BUF.START     POINT TO FIRST BUFFER
A7DD- AE 01 9D      1230         LDX BUF.START+1
                    1240 *-------------------------------
A7E0- 85 40         1250 .1      STA PNTR
A7E2- 86 41         1260         STX PNTR+1
A7E4- A0 00         1270         LDY #0            Store zero over 1st char of
A7E6- 98            1280         TYA               filename to mark it as a
A7E7- 91 40         1290         STA (PNTR),Y      free buffer.
                    1300 *---FILL IN 3 PNTRS--------------
A7E9- 38            1310         SEC               COMPUTE LOW BYTE OF POINTERS
A7EA- A5 40         1320         LDA PNTR
A7EC- E9 2D         1330         SBC #$2D
A7EE- A0 1E         1340         LDY #$1E          ...FMW ADDR
A7F0- 91 40         1350         STA (PNTR),Y
A7F2- A0 20         1360         LDY #$20          ...TSL ADDR
A7F4- 91 40         1370         STA (PNTR),Y
A7F6- A0 22         1380         LDY #$22          ...DATA ADDR
A7F8- 91 40         1390         STA (PNTR),Y
A7FA- 48            1400         PHA
A7FB- A5 41         1410         LDA PNTR+1        COMPUTE HIGH BYTE OF FMW ADDR
A7FD- E9 00         1420         SBC #0
A7FF- A0 1F         1430         LDY #$1F          ...FMW ADDR
A801- 91 40         1440         STA (PNTR),Y
A803- E9 01         1450         SBC #1
A805- A0 21         1460         LDY #$21          ...TSL ADDR
A807- 91 40         1470         STA (PNTR),Y
A809- E9 01         1480         SBC #1
A80B- A0 23         1490         LDY #$23          ...DATA ADDR
A80D- 91 40         1500         STA (PNTR),Y
                    1510 *---IS THAT THE LAST BUFFER?-----
A80F- C8            1520         INY               POINT AT FWD LINK LO-BYTE
A810- AA            1530         TAX            SAVE HI BYTE OF DATA ADDR
A811- CE 63 AA      1540         DEC TEMP
A814- F0 10         1550         BEQ .2         ...NO MORE BUFFERS
                    1560 *---BUILD LINK TO NEXT BUFFER----
A816- 68            1570         PLA            GET LO BYTE
A817- E9 26         1580         SBC #$26       ADDR OF FILENAME IN NEXT BUFFER
A819- 91 40         1590         STA (PNTR),Y      ...LO BYTE
A81B- 48            1600         PHA               SAVE ON STACK
A81C- 8A            1610         TXA               GET HI BYTE
A81D- E9 00         1620         SBC #0
A81F- C8            1630         INY               ...HI BYTE
A820- 91 40         1640         STA (PNTR),Y
A822- AA            1650         TAX               SAVE IN X
A823- 68            1660         PLA               GET LO BYTE AGAIN
A824- B0 BA         1670         BCS .1            ...ALWAYS
                    1680 *---SET FORWARD PNTR = 0000------
A826- A9 00         1690 .2      LDA #0
A828- 91 40         1700         STA (PNTR),Y
A82A- C8            1710         INY
A82B- 91 40         1720         STA (PNTR),Y
                    1730 *---SET HIMEM AND EMPTY BLOCK----
A82D- AD B6 AA      1740         LDA ACTIVE.BASIC.FLAG
A830- F0 0A         1750         BEQ .3            INTEGER BASIC
A832- 86 74         1760         STX FP.HIMEM+1    APPLESOFT
A834- 86 70         1770         STX FP.STRINGS+1
```

```
A836- 68       1780         PLA
A837- 85 73    1790         STA FP.HIMEM
A839- 85 6F    1800         STA FP.STRINGS
A83B- 60       1810         RTS
A83C- 86 4D    1820 .3      STX HIMEM+1
A83E- 86 CB    1830         STX PP+1
A840- 68       1840         PLA
A841- 85 4C    1850         STA HIMEM
A843- 85 CA    1860         STA PP
A845- 60       1870         RTS
               1880 *-------------------------------
```

I found it even more interesting to re-write this program using the 65802 capabilities. The 16-bit registers save a lot of byte shuffling, and eliminate the need for TEMP and PNTR. What's more, instead of saving only 11 bytes over the original DOS 3.3 version, this time I whacked out 46 bytes! And it could be made even smaller, if we could make some assumptions about the CPU status.

In general, we don't know whether we are in 65802 or 6502 mode until we peek at the "hidden" status bit (the E-bit). In the process of peeking we may change it, and may also change the M- and X-bits. Lines 1190 save the current status, flip into '802 mode and save the status again. The first PHP is there in case we were already in '802 mode. If we were, it saves the M- and X- bits and they will be restored by the PLP at line 1620. The second PHP saves the status of the mysterious E-bit (the XCE opcode swaps E and C). Lines 1600-1610 pull this saved status and do another XCE, restoring E to what it was when this sub-routine was called. If we could ASSUME that we were called in '802 mode, we could delete lines 1190-1210 and lines 1610-1620 (saving 5 more bytes). Or, if we could be sure we were always called from 6502 mode, we could delete 1190, 1220, and 1620, and change line 1600 to ".4 SEC" (saving 3 bytes). Probably better never to assume, at least until we are a lot more familiar with this marvelous chip.

The XCE instruction swaps the C- and E-bits, but that is not necessarily all. The M- and X-bits always come up in the 8-bit mode after an XCE. Therefore in line 1240, the LDX will load $00 into the high byte of the X-register and the number of buffers into the low byte. In line 1250 I turn on 16-bit mode for both indexing and memory-accumulator operations, and I will keep it that way until the PLP at line 1600.

6502 programs are always full of page zero pointer addressing modes, but in 65802 programs we may see a lot less of them. Now we can load a whole 16-bit address into the X- or Y-register.

Instead of:	We can write:
LDA BUF.PNTR	
STA PNTR	
LDA BUF.PNTR+1	
STA PNTR+1	
LDY #$1E	LDY BUF.PNTR
LDA DATA...	LDA DATA...
STA (PNTR),Y	STA $1E,Y

Lines 1280-1290 zero the first byte of the filename. As an "extra" feature now, the second byte is also zeroed. In lines 1300-1380 I can compute and store the three area pointers in a very straightforward manner. It now occurs to me that by swapping the roles of the X- and Y-registers I could save six more bytes, since the STA $offset,X instructions would assemble in two bytes rather than three. (The only problem might be that the D-register must = $0000 for this to work.)

Since I don't have to use the X-register to hold temporary values during the buffer creation loop, I can use it instead to count buffers. Lines 1400-1410 do the counting.

If we have not just built the last buffer, lines 1420-1460 set the "next buffer" link address and branch back to build another buffer.

Lines 1470-1500 save the address of the data area in the X-register and store 0000 in the link address for the last buffer. The data area address is going to be the new HIMEM value.

Lines 1510-1590 store the new HIMEM value for the currently selected language. If we are in Applesoft, the string area normally bumps against HIMEM; we now empty that area, because HIMEM may have moved. If we are in Integer BASIS or the S-C Assembler (which fools DOS into believing it is I/B), the source program nestles against HIMEM; it is therefore emptied by storing the HIMEM value into PP.

Won't it be nice when we all have 65802's and can USE these new code segments? It may not be as long as you think. In the mean time, maybe we can develop our expertise. And we can carve enough holes in DOS to leave room for some great new features.

```
                    1000 *SAVE S.INIT BUFFERS (802)
                         1010         .OP 65816
                         1020 *-------------------------------
                         1030 *   REPLACEMENT FOR DOS 3.3 CODE
                         1040 *      (SAVES 46 BYTES, NO CHANGE IN FUNCTION)
                         1050 *-------------------------------
 4C-                     1060 HIMEM              .EQ $4C,4D
 6F-                     1070 FP.STRINGS         .EQ $6F,70
 73-                     1080 FP.HIMEM           .EQ $73,74
 CA-                     1090 PP                 .EQ $CA,CB
                         1100 *-------------------------------
 9D00-                   1110 BUF.START          .EQ $9D00
 AA57-                   1120 NO.FILES           .EQ $AA57
 AAB6-                   1130 ACTIVE.BASIC.FLAG .EQ $AAB6
                         1140 *-------------------------------
                         1150         .OR $A7D4
                         1160         .TA $08D4
                         1170 *-------------------------------
                         1180 INIT.FILE.BUFFERS
 00A7D4- 08              1190         PHP                SAVE CURRENT STATUS AND
 00A7D5- 18              1200         CLC                TURN ON 802 MODE
 00A7D6- FB              1210         XCE
 00A7D7- 08              1220         PHP
                         1230 *-------------------------------
 00A7D8- AE 57 AA        1240         LDX NO.FILES       DO (NO.FILES) TIMES
 00A7DB- C2 30           1250         REP #$30           16-BIT OPERATIONS
 00A7DD- AC 00 9D        1260         LDY BUF.START      POINT TO FIRST BUFFER
                         1270 *-------------------------------
```

```
00A7E0- A9 00 00    1280 .1     LDA ##0           STORE ZERO OVER 1ST & 2ND CHARS
00A7E3- 99 00 00    1290        STA 0,Y           OF FILENAME TO FREE BUFFER
                    1300 *---FILL IN 3 PNTRS--------------
00A7E6- 38          1310        SEC               COMPUTE LOW BYTE OF POINTERS
00A7E7- 98          1320        TYA               FROM FILENAME ADDR
00A7E8- E9 2D 00    1330        SBC ##$2D
00A7EB- 99 1E 00    1340        STA $1E,Y         ...FMW ADDR
00A7EE- E9 00 01    1350        SBC ##$100
00A7F1- 99 20 00    1360        STA $20,Y         ...TSL ADDR
00A7F4- E9 00 01    1370        SBC ##$100
00A7F7- 99 22 00    1380        STA $22,Y         ...DATA ADDR
                    1390 *---IS THAT THE LAST BUFFER?-----
00A7FA- CA          1400        DEX
00A7FB- F0 09       1410        BEQ .2          ...NO MORE BUFFERS
                    1420 *---BUILD LINK TO NEXT BUFFER----
00A7FD- E9 26 00    1430        SBC ##$26     ADDR OF FILENAME IN NEXT BUFFER
00A800- 99 24 00    1440        STA $24,Y
00A803- A8          1450        TAY           BASE ADDRESS FOR NEXT BUFFER
00A804- 80 DA       1460        BRA .1          ...ALWAYS
                    1470 *---SET FORWARD PNTR = 0000------
00A806- AA          1480 .2     TAX           SAVE HIMEM VALUE
00A807- A9 00 00    1490        LDA ##0
00A80A- 99 24 00    1500        STA $24,Y
                    1510 *---SET HIMEM AND EMPTY BLOCK----
00A80D- AD B6 AA    1520        LDA ACTIVE.BASIC.FLAG
00A810- 29 FF 00    1530        AND ##$FF
00A813- F0 06       1540        BEQ .3            INTEGER BASIC
00A815- 86 73       1550        STX FP.HIMEM      APPLESOFT
00A817- 86 6F       1560        STX FP.STRINGS
00A819- 80 04       1570        BRA .4
00A81B- 86 4C       1580 .3     STX HIMEM         INTEGER BASIC
00A81D- 86 CA       1590        STX PP
00A81F- 28          1600 .4     PLP
00A820- FB          1610        XCE
00A821- 28          1620        PLP
00A822- 60          1630        RTS
                    1640 *-------------------------------
```

65C02s in Old Apples.....Jim Sather

I read Andrew Jackson's 12/84 AAL comments on 65C02 operation in an Apple II with interest since I had looked into the same subject while doing research for "Understanding the Apple IIe". I share Mr. Jackson's conclusion that the problem is short read data setup time from motherboard RAM, but I disagree with his analysis and conclusion that a 65C02 only gets a setup of 25 nsec in an Apple II.

The motherboard RAM read data setup time in an Apple II is

```
        70 nsec (one 14M period)
  minus LS174 pin 9 to data out propagation delay (B5/B8 latch)
  minus LS257 data propagation delay (B6/B7 mux)
  minus 8304 or 8T28 data propagation delay (H10/H11 driver)
   plus MPU PHASE 0 to PHASE 2 propagation delay
   plus 74LS08 propagation delay (B11 PHASE 0 gate).
```

Longer PHASE 0 - PHASE 2 delays result in longer read data setup time, not shorter. With the 6502s and 65C02s I have experimented with, PHASE 0 to PHASE 2 delay has always been in the 20-40 nsec region. Whatever the variation, I have found no NCR or GTE 65C02 that will work in my Apple II.

Taking all delays into account, the motherboard read data setup time for a 6502 or 65C02 is about 65 nsec. This is not good enough for 1 MHz 6502/65C02 specifications but it is good enough for 2 MHz 6502/65C02 specifications. In other words, the Apple II does not meet the read data setup spec of the 1 MHz 6502 that it was manufactured with. Based on this fact, the 100 nsec read data setup spec of 1 MHz 6502s is unrealistically conservative.

But why won't a 2 MHz 65C02 run in the Apple II if it requires only 50 nsec setup time and it gets 65 nsec? The answer, in my opinion, is that NCR and GTE 2 MHz 65C02s do not operate to spec. With certain instruction sequences, they require more than 50 (and, in fact, more than 65) nsec read data setup time. The instruction sequences that bomb are VERY limited, so the 65C02 only gets into trouble when a certain few code sequences are executed. The 65C02 symptom in the Apple II is, therefore, that most things work, but some don't.

Efforts to improve 65C02 operation in the Apple II can be concentrated on decreasing data delays (by replacing the LS174s and LS257s with equivalent devices from a faster logic family) or increasing MPU data clock delays (by adding TTL devices in series with the MPU PHASE 0 input). Possible reduction in data delays is limited, so increased MPU PHASE 0 delay is tempting. Be forewarned, though, that 6502 PHASE 2 is already very late for peripheral slot and serial input mux data transfer, and that such data transfer already depends on the long bleed off time of data from the floating data bus. It is certainly feasible that some Apples with heavy data bus loads will begin to show bugs if any MPU PHASE 0 delay is introduced. But in all probability, you can increase the MPU PHASE 0 delay in a given Apple until MPU PHASE 2 falls concurrently with RAM

SELECT' after access to an address above $C00F in the Apple II. This point is 60 nsec after peripheral slot PHASE 0 falls in my Apple II.

AAL readers may be interested in the following excerpt from "Understanding the Apple IIe". It details some features of the 65C02 which are not clear from the data sheet and describes instruction sequences that I have found that make NCR and GTE 65C02s bomb in an Apple II. Note particularly that I have a Rockwell 1 MHz 65C02 that operates without a hitch in my Apple II. This may be a lucky coincidence, or Rockwell 65C02s may not have the read data setup problems of the NCR and GTE chips.

[Following is an excerpt from "Understanding the Apple //e", copyright (c) 1985 by Quality Software, published here by permission of Quality Software.]

THE 65C02 MICROPROCESSOR

A recent development in the 6502 world has been the introduction of the 65C02 MPU. This MPU (manufactured by NCR, Rockwell, and alternate sources) is fabricated using CMOS technology, instead of the NMOS used in the 6502. The general advantage of CMOS over NMOS is lower power consumption, but the 65C02 also has some new instructions which make it operationally more powerful than its NMOS brother. A 65C02 can execute any 6502 program that doesn't depend on fine instruction execution timing, but a 6502 cannot execute 65C02 programs that utilize the new 65C02 instructions.

Apple uses the 65C02 MPU in the Apple //c microcomputer, and they intend to convert the Apple //e over to the 65C02. The plan is to retrofit older Apple //e's with the 65C02 as part of the firmware upgrade package described in Chapter 6. This will maximize compatiblity betweeen the Apple //e and the Apple //c, and make it possible to write shorter and faster Apple //e assembly language programs. Because the Apple //e may become a 65C02 based computer in the future, some data on the 65C02 is given here and in other parts of "Understanding the Apple //e".

The 65C02 improvements consist of the addition of new instructions and addressing modes, and the removal of some old 6502 bugs. For the most part, differences between the 6502 and 65C02 are well documented in the partial NCR 65C02 data sheet in Appendix C at the back of this book. Descriptions here will therefore be limited to a few points whose ramifications are not made entriely clear by the data sheet. Please note also that details of 65C02 instruction execution are given in Tables 4.3 and 4.4 in an application note later in this chapter.

First, the NCR and Rockwell 65C02s are not identical. The Rockwell chip executes some instructions that are not part of the NCR 65C02 repertoire. These are the zero page instructions RMBn (Reset Memory Bit n) and SMBn (Set Memory Bit n), and the zero page relative branch instructions BBRn (Branch on Bit n Reset) and BBSn (Branch on Bit n Set). The opcodes of these Rockwell instructions ($X7 and $XF) represent NOPs in the

NCR chip. Apple appears to be using NCR compatible 65C02s in its computers, but the Rockwell chip works fine in the Apple //e. Please refer to Tables 4.3 and 4.4 for details of the additional Rockwell instructions.

The READY line of a 6502 will not halt the MPU during a write cycle, but the 65C02 READY line will. This raises the question, "what happens to the Apple IIe data bus if READY is pulled low during a write cycle and is held low for a number of following write cycles?" If the 65C02 attempts to control the data bus constantly for a series of wait state write cycles, it will compete with motherboard RAM for control of the data bus near the end of PHASE 1. Investigation shows that this is not a problem. During a long series of wait state write cycles, the 65C02 control the data bus only during that portion of the machine cycle in which it controls the data bus during a normal write cycle. Therefore, its data bus connection is at high impedance during the majority of PHASE 1 in all wait state write cycles, and motherboard RAM is free to control the data bus near the end of PHASE 1.

The fact that interrupts do not cause abortion of a BREAK instruction is listed as an operational enhancement of the 65C02 on page 3 of the data sheet. The data sheet is referring to non-maskable interrupts, not interrupt requests. In a 6502 or 65C02, IRQ' falling after a BREAK op code fetch does not interfere with BREAK execution. However, if NMI' falls after a BREAK op code fetch and before the interrupt vector is fetched in a 6502, then the NMI' interrupt vector is fetched, and the NMI' handler is executed. An RTI at the end of the NMI' handler causes return to the address (plus two) of the BREAK instruction and probable program crashing. This bug is fixed in the 65C02. As the data sheet indicates, NMI' falling during BREAK execution results in NMI' execution after BREAK execution is complete.

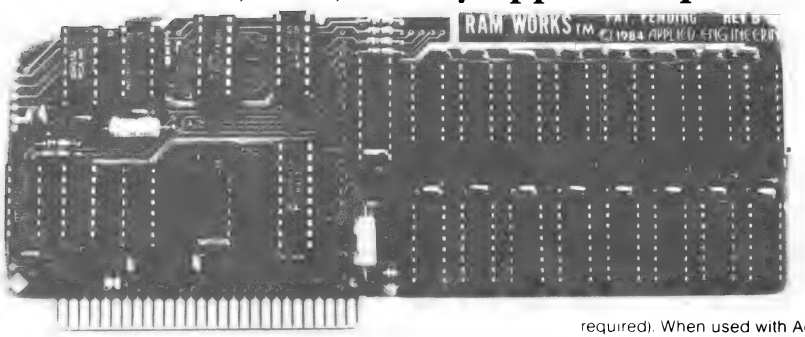
The NCR data sheet refers to the new increment accumulator and decrement accumulator instrucions as INA and DEA. I don't know why they do this, because these instructions are clearly just new addressing modes of the INC and DEC instructions. The new mnemonics should be INC A and DEC A or just INC and DEC as given in the Rockwell data sheet. The addition of the INC and DEC accumulator addressing modes means these instructions have all the addressing modes of the other 6502 read-modify-write instructions (ASL, LSR, ROL, and ROR).

Another notable feature of the 65C02 data sheet is the 5000-microsecond maximum cycle time in the AC characteristics table on page 3. I take this to mean that you can stop the clock for a guaranteed minimum of 5000 microseconds with PHASE 0 high, but not with PHASE 0 low. The Rockwell data sheet is more specific about the difference. It states: "The input clock can be held in the high state indefinitely; however, if the input clock is held in the low state longer than 5 microseconds, internal register and data status can be lost". The significance is that, when the Apple IIe DMA' line is held low, it forces the PHASE 0 input to the MPU to a low state. I therefore conclude that long term continuous DMA in the Apple

RAMWORKS™

800K Apple Works Desk Top, 450K Visicalc, 1 Meg Solid State Disk

Double Hi-Res, RGB, Totally Apple Compatible!



RAMWORKS™—A card that plugs into the Apple IIe auxiliary slot and functions EXACTLY like Apple's extended 80 column card (in fact, a 128K RAMWORKS™ actually costs less than Apple's 64K extended card) but with RAMWORKS™ you get more memory, 80 column text, a 3-year warranty and most importantly, room to grow without using more slots. A design so advanced there's a patent pending on it. If you have a IIc or an IBM, we suggest you do what everybody does, trade it in on a IIe.

RAMWORKS™ can be purchased in a wide range of sizes and is user upgradeable using either 64K RAMS or the new 256K RAMS. In fact, RAMWORKS™ is the only auxiliary slot card on the market that will allow the new 256K RAMS to be used. If you already have an extended 80 column card, no problem. Just unplug the 64K RAMS and plug them into the RAMWORKS™ for an additional 64K. A RGB option is also available, you can order it with your RAMWORKS™ card or add it on at a later date.

RAMWORKS™ saves you time, money, slots and hassle. You'll have additional memory NOW and in the future.

Ramworks™	
64K Installed	$ 179
128K Installed	$ 249
256K Installed	$ 399
512K Installed	$ 649
1 MEG Installed	$1199
RGB Option	$ 129

(May be added later)

LOW COST SOFTWARE OPTIONS

Ram Drive IIe

Ram Drive IIe will give you a high speed solid state disk drive. The Ram Drive IIe software features audio-visual access indicators, easy setup for turnkey operation, and easy menu driven documentation. The program can be modified and is copyable. If you have a 64K RAMWORKS™, Ram Drive IIe will act as half a disk drive. If you have a 128K or larger RAMWORKS™, Ram Drive IIe will act as a full disk drive. Either way, your programs will load and save over 20 times faster. Ram Drive IIe is compatible with APPLESOFT, PRO DOS, DOS 3.3, and PASCAL. The disk also includes a high speed RAMdisk copying program. Ram Drive is another disk drive only 20 times faster. And no whirring, clicking and waiting! **PRICE $29**

CP/M Ram Drive IIe

CP/M Ram Drive IIe is just like the Ram Drive IIe above, only for CP/M.

CP/M Ram Drive IIe runs on any Z-80 card that runs standard CP/M i.e. Applied Engineering Z-80 Plus or Microsoft Soft Card. CP/M Ram Drive will dramatically speed up the operation of most CP/M software because CP/M normally goes to disk fairly often. Fast acting software like dBase II, Wordstar and Turbo Pascal becomes virtually instantaneous when used with CP/M Ram Drive. **PRICE $29**

VC IIe Expander

VC IIe expander gives owners of Visicalc IIe and Advanced Visicalc IIe increased storage. When used with VC IIe you'll get 141K workspace (128K RAMWORKS™ or larger required). When used with Advanced VC IIe you'll get 131K with a 128K RAMWORKS™, 250K with a 256K RAMWORKS™ and 450K with a 512K RAMWORKS™ **PRICE $29**

Apple Works Expand

Although Apple Works is compatible with all sizes of RAMWORKS™, Apple Works only "sees" its first 64K bank giving you a 55K desktop. Our Apple Works expand program will make a modification to Apple Works that simply lets it know you've got more memory, giving you 101K work space. **PRICE $29**

Super Apple Works Expand

This souped-up version of Apple Works expand doesn't stop at a 101K desk top, in fact Super Apple Works Expand figures out how much memory your RAMWORKS™ has to give Apple Works the following desktop sizes:

RAMWORKS™	APPLEWORKS DESKTOP
128K	101K
256K	200K
512K	400K
1 MEG	800K

PRICE $39

AppleWorks can also be put in the RAMWORKS™ card to eliminate disk access, thereby dramatically speeding up the program.

APPLIED ENGINEERING

Send Check or Money Order to:
Applied Engineering
P.O. Box 798 Carrollton, TX 75006
Call (214) 492-2027
8 a.m. to 11 p.m. 7 days a week. MasterCard, Visa & C.O.D. Welcome. No extra charge for credit cards. Texas residents add 5% Sales tax. Add $10.00 if outside U.S.A.

IIe cannot be performed with a 65C02 any easier than it can with a 6502. In either case, long term continuous DMA can only be performed by pulling DMA' low after the MPU has been stopped via READY low, and only after the X4 and X5 Apple IIe motherboard jumpers have been configured so the MPU clock is not stopped when DMA' is pulled low.

A feature of the 65C02 that does not show up in the NCR data sheet is that the new BIT immediate instruction operates differently than BIT in the other addressing modes. In the other addressing modes, BIT sets the negative, overflow, and zero flags based respectively on operand bit 7, operand bit 6, and the result of Accumulator AND operand. The 65C02 BIT immediate instruction affects only the zero flag, not the negative and overflow flags.

A final point about 65C02 operation that I'd like to make is mildly speculative. The 65C02 is pin compatible with the 6502, and was designed as a direct but more powerful substitute for the 6502. To make it work in the Apple IIe, you simply remove the 6502 and plug in the 65C02. However, the 65C02 does not work reliably in the older Apple II. I believe that the reason for this is that the 65C02 (or at least an NCR 65C02) requires read data to be set up longer than a 6502 operating at the same frequency. RAM read data in the Apple II becomes valid at the MPU (about 60 nsec before PHASE 2 falls) much later than it does in the Apple IIe (about 250 nsec before PHASE 2 falls). Whereas the 6502 can handle the short RAM read data set up time, the 65C02 seems to have trouble with it.

I have performed limited experiments with 65C02s in an Apple II. Basically, I found that two NCR 65C02As (2 MHz?) and one NCR compatible GTE G65SC02P-2 (2 MHz) caused intermittent program crashing that got worse as the peripheral card data bus load was increased. The Rockwell R65C02P1 (1 MHz) that I tried caused no program crashes. The NCR 65C02 program drashes occurred only with certain data bus sequences. If an RTS instruction is preceded by a NOP or SBC, and the Apple II video data preceding the RTS opcode fetch is $A0, $A2, or $A9 then the carry flag is set during otherwise normal execution of the RTS instruction. This unwanted setting of the carry flag occurred as mentioned with all three NCR type chips. One of the chips also set the carry flag if the video data preceding the RTS was $89, and another one also set the carry flag if the video data preceding RTS was $89 or $E9. Note that $89, $A0, $A2, $A9, and $E9 are all immediate mode 65C02 instructions.

In these experiments, I did not conclusively prove that the problem with the 65C02 in the Apple II is short set up time of RAM read data. This is merely a highly educated guess upon which I would be willing to bet a paycheck (if only I had one). Setting the data up quicker definitely helps, because the bugs mentioned in the previous paragraph do not exist when the program resides in a 16K RAM card whose read data becomes valid just after Q3 falls during PHASE 0. In any case, I am suspicious of the validity of the NCR claim of 50-nsec minimum read data set up time in its 65C02.

Improved DOS 3.3 Number Parsing.....Bob Sander-Cederlof & Lower-Case DOS Commands

Whether Apple knows or not, cares or not, likes it or not, DOS 3.3 is still alive. And still the system of choice to most of their loyal customers.

The //c and new //e ROMs patch Applesoft so that lower case keywords and commands can be typed without penalty. However, since they are promoting ProDOS and do not care about DOS, they did nothing to give DOS the freedom to accept lower case commands. I am constantly chafing at the necessity of popping the shift lock key up and down, (down for DOS and up for word processing). Surely a very small patch would do the trick.

I looked around and found the subroutine DOS uses to pick characters out of the command buffer, at $A193-$A1AD. Six bytes of new code inserted right before the CMP #$AC at $A1A1 would do it. If I put a JSR to a patch in place of the STX $AA5D at $A19E, a ten-byte patch subroutine would solve my problem.

But where do I get a ten-byte hole to fit this patch into? All the holes I know about have already been used now, and I really don't want to eliminate any existing features. The only solution is to find some loosely written code and rewrite it with compactness as the major criterion.

The code to be recoded must be relatively unused. That is, not likely to be called at internal places by sneaky software. I found a likely candidate in the number conversion subroutine used in parsing DOS commands. This subroutine occupies from $A1B9 through $A228. I ran a cross reference on the outer shell portion of DOS ($9D84-$A883) using Rak-Ware's DISASM program, and verified that there are no entry points into this code except at the beginning. It is called from only two places, $A0AA and $A127.

Here is a commented disassembly of the subroutine:

```
                    1000 *SAVE S.DOS NUMIN
                    1010 *--------------------------------
44-                 1020 NUML   .EQ $44
45-                 1030 NUMH   .EQ $45
                    1040 *--------------------------------
A1A4-               1050 GNNB   .EQ $A1A4
                    1060 *--------------------------------
                    1070        .OR $A1B9
                    1080        .TA $09B9
                    1090 *--------------------------------
                    1100 *      RETURN .CC. WITH NUMBER IN A,X
                    1110 *          OR .CS. IF BAD SYNTAX
                    1120 *--------------------------------
                    1130 CONVERT.NUMBER.IN.WBUF
A1B9- A9 00         1140        LDA #0      INIT NUMBER = 0
A1BB- 85 44         1150        STA NUML
A1BD- 85 45         1160        STA NUMH
A1BF- 20 A4 A1      1170        JSR GNNB    GET NEXT NON-BLANK CHAR
A1C2- 08            1180        PHP
A1C3- C9 A4         1190        CMP #"$"    HEX OR DECIMAL?
A1C5- F0 3C         1200        BEQ .6      ...HEX
A1C7- 28            1210        PLP
A1C8- 4C CE A1      1220        JMP .2      ...DECIMAL (OR NONE)
                    1230 *---NEXT CHAR OF DECIMAL #-------
A1CB- 20 A4 A1      1240 .1     JSR GNNB    GET NEXT NON-BLANK CHAR
A1CE- D0 06         1250 .2     BNE .3      ...NOT COMMA OR CR
A1D0- A6 44         1260        LDX NUML    END OF NUMBER
```

```
A1D2- A5 45     1270         LDA NUMH      VALUE IN A,X
A1D4- 18        1280         CLC           SIGNAL VALID NUMBER
A1D5- 60        1290         RTS           RETURN
                1300 *---CONVERT DECIMAL NUMBER-------
A1D6- 38        1310 .3      SEC           CONVERT CHAR TO DIGIT
A1D7- E9 B0     1320         SBC #$B0
A1D9- 30 21     1330         BMI .4        ...NOT DIGIT
A1DB- C9 0A     1340         CMP #$0A
A1DD- B0 1D     1350         BCS .4        ...NOT DIGIT
A1DF- 20 FE A1  1360         JSR .5        SHIFT VALUE 1 LEFT
A1E2- 65 44     1370         ADC NUML      2*VALUE + DIGIT
A1E4- AA        1380         TAX
A1E5- A9 00     1390         LDA #$00
A1E7- 65 45     1400         ADC NUMH
A1E9- A8        1410         TAY
A1EA- 20 FE A1  1420         JSR .5        SHIFT VALUE 1 LEFT
A1ED- 20 FE A1  1430         JSR .5        SHIFT VALUE 1 LEFT
A1F0- 8A        1440         TXA           ...+ 8*VALUE
A1F1- 65 44     1450         ADC NUML
A1F3- 85 44     1460         STA NUML
A1F5- 98        1470         TYA
A1F6- 65 45     1480         ADC NUMH
A1F8- 85 45     1490         STA NUMH
A1FA- 90 CF     1500         BCC .1        ...NO OVERFLOW
A1FC- 38        1510 .4      SEC           SIGNAL BAD CHAR OR OVERFLOW
A1FD- 60        1520         RTS
                1530 *---SHIFT VALUE 1 BIT LEFT-------
A1FE- 06 44     1540 .5      ASL NUML
A200- 26 45     1550         ROL NUMH
A202- 60        1560         RTS
                1570 *---CONVERT HEX NUMBER-----------
A203- 28        1580 .6      PLP           POP USELESS STATUS
A204- 20 A4 A1  1590 .7      JSR GNNB      GET NEXT NON-BLANK CHAR
A207- F0 C5     1600         BEQ .2        ...END OF NUMBER
A209- 38        1610         SEC           CONVERT ASCII TO DIGIT
A20A- E9 B0     1620         SBC #$B0
A20C- 30 EE     1630         BMI .4        ...NOT A DIGIT
A20E- C9 0A     1640         CMP #$0A
A210- 90 08     1650         BCC .8        ...0-9
A212- E9 07     1660         SBC #$07      TRY LETTERS
A214- 30 E6     1670         BMI .4        ...NOT A DIGIT
A216- C9 10     1680         CMP #$10
A218- B0 E2     1690         BCS .4        ...NOT A DIGIT
A21A- A2 04     1700 .8      LDX #4        SHIFT VALUE 4 BITS LEFT
A21C- 20 FE A1  1710 .9      JSR .5        SHIFT VALUE 1 LEFT
A21F- CA        1720         DEX
A220- D0 FA     1730         BNE .9
A222- 05 44     1740         ORA NUML      MERGE VALUE WITH NEW DIGIT
A224- 85 44     1750         STA NUML
A226- 4C 04 A2  1760         JMP .7        ...NEXT DIGIT
```

Lines 1530-2120 of the following listing shows my revised version, which is sixteen bytes shorter. It is also a little faster, though that is not important. As far as I can tell, no features are changed. There is room for my ten-byte lower-case patch and six bytes to spare!

Compare the two versions to see where I found the extra bytes. Part of the savings was gained by using a better algorithm for reducing an ASCII character to a hex or decimal digit. Changing the order of the sections of the program saved more bytes, by eliminating JMPs and "branch always" ops. I kept the same local labels in the new version to aid you in locating similar sections.

It is always nice to be able to make a self-installing patch, so I dug out the April 83 issue of AAL for Bill Morgan's PATCHER program. I found the source on a Quarterly Disk, and merged it with the new number parser. Then I added my lower case patch, and glued it all together. The listing that follows is the result. If the program is BRUN it will install the new parser and the lower case filter automatically.

```
                        1000 *SAVE S.DOS LC PATCHES
                        1010 *-------------------------------
00-                     1020 PNTR    .EQ $00,01
02-                     1030 PATCH   .EQ $02,03
                        1040 *-------------------------------
                        1050         .OR $300
                        1060         .TF B.DOS LC PATCHES
                        1070 *-------------------------------
                        1080 PATCHER
0300- A9 33             1090         LDA #PATCHES-1
0302- 85 00             1100         STA PNTR
0304- A9 03             1110         LDA /PATCHES-1
0306- 85 01             1120         STA PNTR+1
0308- A0 00             1130         LDY #0
                        1140
030A- 20 2B 03          1150 .1      JSR GET.BYTE LENGTH OF NEXT PATCH
030D- F0 1B             1160         BEQ .4       FINISHED
030F- AA                1170         TAX          SAVE LENGTH IN X
0310- 20 2B 03          1180         JSR GET.BYTE ADDRESS OF PATCH
0313- 85 02             1190         STA PATCH
0315- 20 2B 03          1200         JSR GET.BYTE
0318- 85 03             1210         STA PATCH+1
                        1220
031A- 20 2B 03          1230 .2      JSR GET.BYTE
031D- 91 02             1240         STA (PATCH),Y
031F- E6 02             1250         INC PATCH
0321- D0 02             1260         BNE .3
0323- E6 03             1270         INC PATCH+1
0325- CA                1280 .3      DEX
0326- D0 F2             1290         BNE .2
0328- F0 E0             1300         BEQ .1     ...ALWAYS
                        1310
032A- 60                1320 .4      RTS
                        1330 *-------------------------------
                        1340 GET.BYTE
032B- E6 00             1350         INC PNTR
032D- D0 02             1360         BNE .1
032F- E6 01             1370         INC PNTR+1
0331- B1 00             1380 .1      LDA (PNTR),Y
0333- 60                1390         RTS
                        1400 *-------------------------------
44-                     1410 NUML    .EQ $44
45-                     1420 NUMH    .EQ $45
                        1430 *-------------------------------
A1A4-                   1440 GNNB    .EQ $A1A4
                        1450 *-------------------------------
                        1460 PATCHES
0334- 70 B9 A1          1470         .DA #P1.LENGTH,$A1B9
                        1480         .PH $A1B9
                        1490 *-------------------------------
                        1500 *       RETURN .CC. WITH NUMBER IN A,X
                        1510 *          OR .CS. IF BAD SYNTAX
                        1520 *-------------------------------
                        1530 CONVERT.NUMBER.IN.WBUF
A1B9- A0 00             1540         LDY #0       INIT NUMBER = 0
A1BB- 84 44             1550         STY NUML     (AND LEAVE Y=0 TOO)
A1BD- 84 45             1560         STY NUMH
A1BF- 20 A4 A1          1570         JSR GNNB     GET NEXT NON-BLANK CHAR
A1C2- F0 2E             1580         BEQ .2       ...NO NUMBER, RETURN 0
A1C4- C9 A4             1590         CMP #"$"     HEX OR DECIMAL?
A1C6- F0 3E             1600         BEQ .7       ...HEX
                        1610 *---CONVERT DECIMAL NUMBER-------
A1C8- 49 B0             1620 .3      EOR #$B0     CONVERT CHAR TO DIGIT
A1CA- C9 0A             1630         CMP #10
A1CC- B0 49             1640         BCS .4       ...NOT DIGIT
A1CE- 06 44             1650         ASL NUML     SHIFT VALUE 1 LEFT
A1D0- 26 45             1660         ROL NUMH
A1D2- 65 44             1670         ADC NUML     2*VALUE + DIGIT
A1D4- AA                1680         TAX
A1D5- 98                1690         TYA          A = Y = 0
A1D6- 65 45             1700         ADC NUMH
A1D8- 48                1710         PHA
A1D9- 06 44             1720         ASL NUML     SHIFT VALUE 1 LEFT
A1DB- 26 45             1730         ROL NUMH
A1DD- 06 44             1740         ASL NUML     SHIFT VALUE 1 LEFT
A1DF- 26 45             1750         ROL NUMH
A1E1- 8A                1760         TXA          ...+ 8*VALUE
A1E2- 65 44             1770         ADC NUML
A1E4- 85 44             1780         STA NUML
```

```
A1E6- 68          1790         PLA
A1E7- 65 45       1800         ADC NUMH
A1E9- 85 45       1810         STA NUMH
A1EB- B0 2A       1820         BCS .4         ...OVERFLOW
A1ED- 20 A4 A1    1830 .1      JSR GNNB       GET NEXT NON-BLANK CHAR
A1F0- D0 D6       1840         BNE .3         ...NOT COMMA OR CR
                  1850 *---NUMBER IS FINISHED----------
A1F2- A6 44       1860 .2      LDX NUML       END OF NUMBER
A1F4- A5 45       1870         LDA NUMH       VALUE IN A,X
A1F6- 18          1880         CLC            SIGNAL VALID NUMBER
A1F7- 60          1890         RTS            RETURN
                  1900 *---MERGE NEXT HEX DIGIT---------
A1F8- 0A          1910 .8      ASL            POSITION DIGIT
A1F9- 0A          1920         ASL
A1FA- 0A          1930         ASL
A1FB- 0A          1940         ASL
A1FC- A2 04       1950         LDX #4         SHIFT VALUE 4 BITS LEFT
A1FE- 0A          1960 .9      ASL            SHIFT DIGIT INTO VALUE
A1FF- 26 44       1970         ROL NUML
A201- 26 45       1980         ROL NUMH
A203- CA          1990         DEX
A204- D0 F8       2000         BNE .9
                  2010 *---CONVERT HEX NUMBER-----------
A206- 20 A4 A1    2020 .7      JSR GNNB       GET NEXT NON-BLANK CHAR
A209- F0 E7       2030         BEQ .2         ...END OF NUMBER
A20B- 49 B0       2040         EOR #$B0       CONVERT ASCII TO DIGIT
A20D- C9 0A       2050         CMP #10        0...9?
A20F- 90 E7       2060         BCC .8         ...YES, 0-9
A211- 69 88       2070         ADC #$88       SHIFT RANGE FOR A-F TEST
A213- C9 FA       2080         CMP #$FA       A...F?
A215- B0 E1       2090         BCS .8         ...A-F
                  2100 *---SYNTAX ERROR-----------------
A217- 38          2110 .4      SEC            SIGNAL BAD CHAR OR OVERFLOW
A218- 60          2120         RTS
                  2130 *--------------------------------
                  2140 GNC.LC.PATCH
A219- 8E 5D AA    2150         STX $AA5D
A21C- C9 E0       2160         CMP #$E0
A21E- 90 02       2170         BCC .1
A220- 29 DF       2180         AND #$DF
A222- 60          2190 .1      RTS
                  2200 *--------------------------------
A223-             2210         .BS $A229-*
                  2220 *--------------------------------
70-               2230 P1.LENGTH .EQ *-$A1B9
                  2240         .EP
                  2250 *--------------------------------
03A7- 03 9E A1    2260         .DA #3,$A19E
                  2270         .PH $A19E
A19E- 20 19 A2    2280         JSR GNC.LC.PATCH
                  2290         .EP
                  2300 *--------------------------------
03AD- 00          2310         .DA #0         END OF PATCHES
```

The Oki 6203 Multiply/Divide Chip.....Bob Sander-Cederlof

If you really need to multiply or divide in a hurry, the Oki 6203 may the ticket. This device sells for about $7, and can be almost directly connected to the Apple bus. All you need is one inverter and a prototyping board.

Assuming you built a little card with the device on it, with its two address lines connected to Apple's A0 and A1 lines, you could multiply two 8-bit numbers for a 16-bit product like this:

```
MUL.6203 STA SLOT*16+$C080   1ST OPERAND
         STY SLOT*16+$C081   2ND OPERAND
         LDA #2              MULTIPLY COMMAND
         STA SLOT*16+$C083   COMMAND REGISTER
         NOP                 DELAY FOR RESULT
         LDA SLOT*16+$C081   HI-BYTE OF PRODUCT
         LDY SLOT*16+$C082   LO-BYTE OF PRODUCT
         RTS
```

A very similar program can divide a 16-bit value by an 8-bit value, producing a quotient and remainder. The time for the multiply is only 22 cycles (plus the JSR and RTS if you make a subroutine), and 24 cycles for the divide.

(Please don't order the chip from us, because we don't sell chips.)

A Disassembler for the 65816.....Bob Sander-Cederlof

When I first got my Apple, there were no books around for learning 6502 assembly language. It took me about 3 months to locate and buy a copy of the 6502 programmer's manual from MOS Technology. About the same time I found a book by William Barden that briefly covered the 8080, 6800, and 6502. But the way I really learned the 6502 was by using Woz's L command in the Apple monitor.

Of course there were no printers or printer interfaces around in those days either, so I spent hours upon hours copying 20 lines at a time off the screen. I wrote down a lot of the monitor, and all of the floating point package and Sweet-16 from the tail end of the Integer BASIC ROM. Fortunately, Apple has never gotten around to eliminating the fabulous L-command from the monitor.

In fact, they have even augmented it. The //c version includes patches to allow disassembly of the additional opcodes and address modes of the 65C02. Since Rak-Ware's DISASM calls on the ROM disassembler to decipher each line of code, the //c version automatically grows to accomodate the 65C02.

Now, what about the 65802 and 65816? It's about time someone wrote a disassembler for that. Someone? Why not me?

It's not easy. On the one hand there is the pressure of competition. Woz's code is SO compact! On the other hand, the new chip is SO complex! It is even ambiguous. There is absolutely no way for a 65816 disassembler to know whether an immediate-mode instruction is two or three bytes long. Only by executing the programming, and tracing it line-by-line, can we tell. And even then, it is possible that a tricky programmer might set up code so that it can be interpreted both ways, depending on other conditions.

To make a long story a little shorter, I did it. You guessed that of course. My solution to the ambiguity problem was to put the burden on the person using it. My solution to the complexity problem was to use extensive tables. My solution to the competition with Woz was to do my best and let him keep his well-deserved glory.

In fact, I started by carefully analyzing Woz's code. The trail starts at $FE9E in the monitor ROM. That short piece of code calls INSTDSP at $F8D0 twenty times to disassemble 20 lines of code. If you take a peek ahead to my listing, lines 1390-1400 patch the language card copy of the monitor inside the L-command loop, so that instead of calling $F8D0 twenty times it calls my disassembler at $0B67 twenty times. (If you are using the language card version of the S-C Macro Assembler, there is a copy of the monitor in the language card too.)

BRUNning the 65816 disassembler will install this little patch and toggle the immediate-mode size flag. Thereafter each 800G command will toggle the state of the immediate-mode size flag. In one state this flag causes immediate mode instructions to be

disassembled as 2-byte instructions; in the other, 3-byte instructions.

The tables are quite complicated, and difficult to type in accurately. Therefore I used macros and let the S-C Macro Assembler do the dirty work. The first table starts at line 1500, and consists of the packed names of the single byte opcodes. The macro at lines 1210-1290 defines how the packing is done. The calling line is of the form ">ON A,B,C,D" where the A, B, and C parameters are the three letters of the opcode name. The D parameter is the letter "A" on those opcodes which might also be multiple-byte: ASL, DEC, INC, LSR, ROR, and ROL.

The packing algorithm is almost the same as the one Woz used in the monitor. Each character is represented by five bits, so that three letters take only 15 bits. The macro sets L1, L2, and L3 to the ASCII value (less 64) of the letters of the opcode name. The .SE directive is used for this so that each invocation of the macro can redefine these variables. This compresses the letters from the range $41...5A to $01...1A. Then the .DA line uses multiplication and addition to pack up the compressed letters. Since arithmetic expressions are parsed by the S-C Macro Assembler in a strict left-to-right fashion, "L1*32+L2*32+L3*2" packs them together.

The "ON" macro also generates a label for the opcode name value by using the opcode name, together with the 4th parameter when present. These names are referred to by another table later on.

The second table is just like the first, but with the names of the longer opcodes instead. Notice that ASL, DEC, etc are in this table too, but without the 4th parameter.

The third and fourth tables have 256 entries, one for every possible opcode byte. Each entry is only one byte long, so each table is 256 bytes. Woz used several smaller tables, because the 6502 didn't use every possible opcode value. The 65816 does define an opcode name for every possible value.

The OPINDEX table uses two macros: "OXA" for single byte opcodes, and "OXB" for longer opcodes. Each entry is a pointer to the name in the OPNAMES.A or OPNAMES.B tables. The pointer is divided by two, leaving room for a flag bit which tells which of the two tables the name is in.

The entries in the OPFORMAT table are offsets into the FMTBL. These are all multiples of 2, because the FMTBL entries are two bytes each.

FMTBL contains coded information indicating how many bytes comprise the instruction and operand, and what the address mode looks like in assembly language. The length can be from two to four bytes, and is coded as 1...3 in the last two bits. The rest of the bits tell which special characters to print and where to print the value of the operand bytes. Single byte opcodes don't have any entries in this table.

One more table, the last one: FMTSTR. This defines the meaning of the bits in FMTBL. Note that the characters are the same as the ones in the various comment lines within FMTBL, only in reverse order.

Finally, we get to the code. The 20-line disassembler calls INSTDSP at line 6180. This starts by calling INSDS1 at line 5760. INSDS1 and INSDS2 are kept as defined points because other software sometimes calls these two points. If you wanted to modify Rak-Ware's DISASM, for example, you would probably need these.

Lines 5760-5840 print the address of the next opcode, and "- ". Lines 5850-5860 pick up that opcode byte. If you enter at INSDS2, have the opcode byte already in the A-register. Lines 5870-5980 dig into the tables to get the opcode name, format, and length for single-byte opcodes. Lines 6000-6160 do the same for longer opcodes. The differences for longer opcodes are several: the second opname table is used, the format is gotten from the tables, and the immediate-mode size flag is used to determine the length of immediate mode opcodes.

Lines 6200-6300 print out the 1-4 bytes of the opcode in hex. If there are less than four bytes, enough blanks are printed so that we always end up in the same position. Lines 6310-6400 unpack the opcode name and print it out. If the opcode is single byte, lines 6410-6420 find out and send us back home (we are finished with this line).

Lines 6430-6450 test the format to detect MVP, MVN, and relative address mode instructions. These special cases are handled by lines 6690-7050. All other operand formats are handled by lines 6470-6680. I see now that I could have put lines 6470-6480 back before line 6430, so that the blank separating the opname from the operand was printed before splitting on the mode. Then lines 6700-6710 could be deleted, saving five bytes. Of course line 6720 would then receive the ".9" label.

Lines 6500-6520 shift out one bit at a time of the format bit string. The corresponding index counts down in the X-register from 10 to 0, and picks a format character from FMTSTR to print. After the character is printed, two special cases are looked for. If the character was "#", meaning immediate mode, and if the immediate-mode size flag indicates long immediates, another "#" is printed. If the character was "$", it is time to print the operand in hex, as two, four, or six digits (lines 6620-6650).

Relative addresses may be either 8-bit or 16-bit. Lines 6780-6820 start the computation for 8-bit values, and call on a monitor routine to finish the printing. Lines 6840-6950 do the same for 16-bit relatives. (There are no two-bit relatives here, no matter what the family tree has borne.)

Finally, lines 6970-7050 print out the two bank bytes for the MVP and MVN instructions. This is different from the way you write MVP and MVN for assembly by the S-C Macro Assembler. In

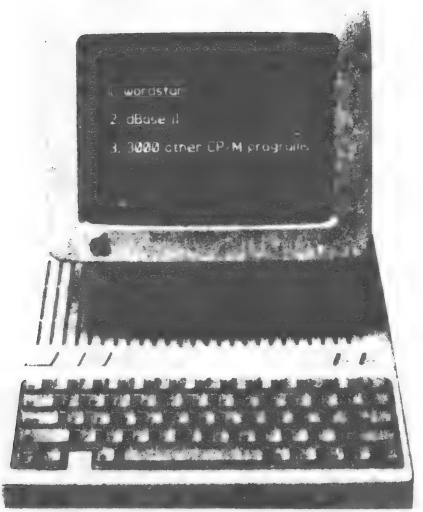
CP/M
for the
APPLE IIc

Apple IIc owners can now run the single largest body of software in existence. Enter the CP/M world with the new Z-80c from Applied Engineering and introduce your IIc to thousands of new programs. With the Z-80c, you'll be able to run Wordstar, dBase II, Turbo Pascal and thousands of other programs that require CP/M. But your IIc will still be your IIc because the Z-80c only turns on when using a CP/M disk.

The Z-80c comes standard with the new 4.0 software, the most advanced system for running CP/M programs ever. But it also runs other versions of CP/M including the popular 2.2 and 2.23 and is fully compatible with Microsoft disks with no pre-boot.

But naturally you'd expect top performance from the Z-80c. After all, it's from Applied Engineering and we've got years of experience with Z-80 cards for the II+ and IIe.

The Z-80c can use all 128K in the IIc, or 64K can be reserved as a RAM disk.



The Z-80c fits neatly inside the IIc. Installation is easy and only takes about 10 minutes (it just plugs in). After installation, your IIc will act and look just like other IIc's, only now you can run all that great CP/M software that others can only dream about.

The Z-80c comes complete with Z-80 card, 4.0 operating system, utility disk, a plain English owners manual and a 3 YEAR WARRANTY. And although the Z-80c is the only CP/M interface on the market for the IIc, we've priced it as though the competition was fierce.

When you consider the fact that many people spend 2 or 3 thousand dollars on a CP/M only computer, our price of $159 is an offer that's practically irresistible.

The Z-80c will give you two computers in one and the advantages of both, all at an unbelievably low price.

Call (214) 492-2027
9 a.m. to 11 p.m.
7 days a week
Send check or money order
P. O. Box 798
Carrollton, Texas 75006

MasterCard, Visa and C.O.D. welcome. No extra charge for credit cards.
Texas residents add 5⅛% sales tax.
Add $10.00 if outside U.S.A.

APPLIED ENGINEERING

the assembler you write "MVP addr1,addr2", where both addresses are 24-bit values. The bank bytes come from the high byte of each 24-bit address. To be compatible with the assembler I should change lines 6970-7050 to print out "0000" after each bank byte.

It seems like a worthy project for someone to incorporate my program into Rak-Ware's DISASM, or perhaps a new similar product. If so, that someone should figure out a neat interactive way to control the immediate-mode size flag. How about it, Bob?

```
                    1010 *SAVE S.65816 DISASM
                    1020 *-------------------------------
00-                 1030 IMM.SIZE .EQ $00
2C-                 1040 LMNEM    .EQ $2C
2D-                 1050 RMNEM    .EQ $2D
2E-                 1060 FORMATL  .EQ $2E
2F-                 1070 LENGTH   .EQ $2F
30-                 1080 FORMATH  .EQ $30
3A-                 1090 PCL      .EQ $3A
3B-                 1100 PCH      .EQ $3B
                    1110 *-------------------------------
F879-               1120 SCRN2  .EQ $F879
F938-               1130 RELADR .EQ $F938
F941-               1140 PRNTAX .EQ $F941
F948-               1150 PRBLNK .EQ $F948
F94A-               1160 PRBL2  .EQ $F94A
F953-               1170 PCADJ  .EQ $F953
FD8E-               1180 CROUT  .EQ $FD8E
FDDA-               1190 PRBYTE .EQ $FDDA
FDED-               1200 COUT   .EQ $FDED
                    1210 *-------------------------------
                    1220         .MA ON
                    1230         .LIST OFF
                    1240 L1      .SE ']1-64
                    1250 L2      .SE ']2-64
                    1260 L3      .SE ']3-64
                    1270         .LIST ON
                    1280 ]1]2]3]4 .DA L1*32+L2*32+L3*2
                    1290         .EM
                    1300 *-------------------------------
                    1310         .MA OXA
                    1320         .DA #]1-OPNAMES.A/2+128
                    1330         .EM
                    1340 *-------------------------------
                    1350         .MA OXB
                    1360         .DA #]1-OPNAMES.B/2
                    1370         .EM
                    1380 *-------------------------------
0800- AD 83 C0      1390 T       LDA $C083
0803- AD 83 C0      1400         LDA $C083
0806- A9 67         1410         LDA #INSTDSP
0808- 8D 65 FE      1420         STA $FE65
080B- A9 0B         1430         LDA /INSTDSP
080D- 8D 66 FE      1440         STA $FE66
0810- A5 00         1450         LDA IMM.SIZE
0812- 49 FF         1460         EOR #$FF
0814- 85 00         1470         STA IMM.SIZE
0816- 60            1480         RTS
                    1490 *-------------------------------
                    1500 OPNAMES.A
0817-               1510         >ON A,S,L,A

0817- D8 0C         0000> ASLA .DA L1*32+L2*32+L3*2
0819-               1520         >ON B,R,K

0819- 96 14         0000> BRK .DA L1*32+L2*32+L3*2
081B-               1530         >ON C,L,C

081B- 06 1B         0000> CLC .DA L1*32+L2*32+L3*2
```

That's enough of that. The assembly listing of that table expands to about 4 pages, so here's a hex dump of OPNAMES.A and OPNAMES.B (By the way, OPNAMES.B .EQ $881):

```
0817- D8
0818- 0C 96 14 06 1B 08 1B 12
0820- 1B 2C 1B E0 1B 46 21 70
0828- 21 72 21 86 4B B0 4B B2
0830- 4B E4 64 E0 73 02 82 04
0838- 82 08 82 16 82 20 82 30
0840- 82 32 82 02 83 04 83 08
0848- 83 20 83 30 83 32 83 D8
0850- 93 E4 93 12 95 18 95 26
0858- 95 46 99 48 99 52 99 20
0860- 9D 70 A0 72 A0 C8 A0 E6
0868- A0 06 A1 C6 A4 F0 A4 02
0870- A6 26 A6 32 A6 42 A6 70
0878- A6 52 B8 1A B9 82 C0 CA
0880- C0 06 09 88 0B D8 0C C6
0888- 10 E6 10 62 11 68 12 52
0890- 13 8A 13 18 14 82 14 98
0898- 14 86 15 A6 15 60 1B 30
08A0- 1C 32 1C 46 21 E4 2B 86
08A8- 4B 58 53 60 53 D8 54 E4
08B0- 54 02 61 30 61 32 61 E4
08B8- 64 9C 6D A0 6D 82 7C 42
08C0- 81 52 81 64 81 60 91 D8
08C8- 93 E4 93 86 98 60 99 02
08D0- 9D 30 9D 32 9D 34 9D 84
08D8- A4 C4 A4
```

```
                    2510 *--------------------------------
                    2520 OPINDEX
                    2530 *---0X----------------------------
08DB-               2540          >0XA BRK
08DB- 81            0000>          .DA #BRK-OPNAMES.A/2+128
08DC-               2550          >0XB ORA
08DC- 1E            0000>          .DA #ORA-OPNAMES.B/2
08DD-               2560          >0XA COP
08DD- 86            0000>          .DA #COP-OPNAMES.A/2+128
```

And the OPINDEX table runs about 7 pages, so another hex dump:

```
08DB- 81 1E 86 1E 2C
08E0- 1E 02 1E 93 1E 80 91 2C
08E8- 1E 02 1E 09 1E 1E 1E 2B
08F0- 1E 02 1E 82 1E 8A A8 2B
08F8- 1E 02 1E 17 01 16 01 06
0900- 01 23 01 99 01 9C 98 06
0908- 01 23 01 07 01 01 01 06
0910- 01 23 01 A1 01 87 AA 06
0918- 01 23 01 9E 12 B2 12 1D
0920- 12 1B 12 8F 12 8D 92 15
0928- 12 1B 12 0C 12 12 12 1C
0930- 12 1B 12 84 12 95 A7 15
0938- 12 1B 12 A0 00 21 00 2A
0940- 00 24 00 96 00 9D 9F 15
0948- 00 24 00 0D 00 00 00 2A
0950- 00 24 00 A3 00 9B A9 15
0958- 00 24 00 0A 27 0B 27 29
0960- 27 28 27 AF 06 AC 90 29
0968- 27 28 27 03 27 27 27 29
0970- 27 28 27 AF 27 AD AE 2A
0978- 27 2A 27 1A 18 19 18 1A
0980- 18 19 18 A6 18 A5 97 1A
0988- 18 19 18 04 18 18 18 1A
0990- 18 19 18 85 18 AB B0 1A
0998- 18 19 18 10 0E 22 0E 10
09A0- 0E 11 0E 8C 0E 88 B1 10
09A8- 0E 11 0E 08 0E 0E 0E 20
09B0- 0E 11 0E 83 0E 94 A4 14
09B8- 0E 11 0E 0F 25 26 25 19
09C0- 25 13 25 8B 25 8E B3 0F
09C8- 25 13 25 05 25 25 25 1F
09D0- 25 13 25 A2 25 9A B4 17
09D8- 25 13 25
```

The assembly listing of OPFORMAT is around a page and a half, so we'll just LIST this one:

```
5250 *--------------------------------
5260 OPFORMAT
5270 F.0     .HS 00.14.00.1C.02.02.02.20.00.00.00.00.04.04.04.06
5280 F.1     .HS 26.16.12.1E.02.08.08.22.00.10.00.00.04.0A.0A.0C
5290 F.2     .HS 04.14.06.1C.02.02.02.20.00.00.00.00.04.04.04.06
5300 F.3     .HS 26.16.12.1E.08.08.08.22.00.10.00.00.0A.0A.0A.0C
5310 F.4     .HS 00.14.00.1C.24.02.02.20.00.00.00.00.04.04.04.06
5320 F.5     .HS 26.16.12.1E.24.08.08.22.00.10.00.00.06.0A.0A.0C
5330 F.6     .HS 00.14.28.1C.02.02.02.20.00.00.00.00.18.04.04.06
5340 F.7     .HS 26.16.12.1E.08.08.08.22.00.10.00.00.1A.0A.0A.0C
5350 F.8     .HS 26.14.28.1C.02.02.02.20.00.00.00.00.04.04.04.06
5360 F.9     .HS 26.16.12.1E.08.08.0E.22.00.10.00.00.04.0A.0A.0C
5370 F.A     .HS 00.14.00.1C.02.02.02.20.00.00.00.00.04.04.04.06
5380 F.B     .HS 26.16.12.1E.08.08.0E.22.00.10.00.00.0A.0A.10.0C
5390 F.C     .HS 00.14.00.1C.02.02.02.20.00.00.00.00.04.04.04.06
5400 F.D     .HS 26.16.12.1E.02.08.08.22.00.10.00.00.18.0A.0A.0C
5410 F.E     .HS 00.14.00.1C.02.02.02.20.00.00.00.00.04.04.04.06
5420 F.F     .HS 26.16.12.1E.08.08.08.22.00.10.00.00.1A.0A.0A.0C
```

```
                        5430 *-------------------------------
                        5440 FMTBL
                        5450 *------# > ( $ , X S ) , Y $ - - - LL
0ADB- 01 90             5460  .DA %1.0.0.1.0.0.0.0.0.0.0.0.0.01 -- IMMEDIATE     00
0ADD- 01 10             5470  .DA %0.0.0.1.0.0.0.0.0.0.0.0.0.01 -- DIRECT        02
0ADF- 02 10             5480  .DA %0.0.0.1.0.0.0.0.0.0.0.0.0.10 -- ABS           04
0AE1- 03 10             5490  .DA %0.0.0.1.0.0.0.0.0.0.0.0.0.11 -- LONG          06
                        5500 *------# > ( $ , X S ) , Y $ - - - LL
0AE3- 01 1C             5510  .DA %0.0.0.1.1.1.0.0.0.0.0.0.0.01 -- DIRECT,X     08
0AE5- 02 1C             5520  .DA %0.0.0.1.1.1.0.0.0.0.0.0.0.10 -- ABS,X        0A
0AE7- 03 1C             5530  .DA %0.0.0.1.1.1.0.0.0.0.0.0.0.11 -- LONG,X       0C
                        5540 *------# > ( $ , X S ) , Y $ - - - LL
0AE9- 41 18             5550  .DA %0.0.0.1.1.0.0.0.0.1.0.0.0.01 -- DIRECT,Y     0E
0AEB- 42 18             5560  .DA %0.0.0.1.1.0.0.0.0.1.0.0.0.10 -- ABS,Y        10
                        5570 *------# > ( $ , X S ) , Y $ - - - LL
0AED- 01 31             5580  .DA %0.0.1.1.0.0.0.1.0.0.0.0.0.01 -- IND          12
0AEF- 01 3D             5590  .DA %0.0.1.1.1.1.1.1.0.0.0.0.0.01 -- INDX         14
0AF1- C1 31             5600  .DA %0.0.1.1.0.0.0.1.1.1.0.0.0.01 -- INDY         16
                        5610 *------# > ( $ , X S ) , Y $ - - - LL
0AF3- 02 31             5620  .DA %0.0.1.1.0.0.0.1.0.0.0.0.0.10 -- INDABS       18
0AF5- 02 3D             5630  .DA %0.0.1.1.1.1.1.1.0.0.0.0.0.10 -- INDABSX      1A
                        5640 *------# > ( $ , X S ) , Y $ - - - LL
0AF7- 01 1A             5650  .DA %0.0.0.1.1.0.1.0.0.0.0.0.0.01 -- STK          1C
0AF9- C1 3B             5660  .DA %0.0.1.1.1.0.1.1.1.1.0.0.0.01 -- STKY         1E
                        5670 *------# > ( $ , X S ) , Y $ - - - LL
0AFB- 01 71             5680  .DA %0.1.1.1.0.0.0.1.0.0.0.0.0.01 -- INDLONG      20
0AFD- C1 71             5690  .DA %0.1.1.1.0.0.0.1.1.1.0.0.0.01 -- INDLONGY     22
0AFF- A2 10             5700  .DA %0.0.0.1.0.0.0.0.0.1.0.1.0.10 -- MVN & MVP    24
0B01- 21 00             5710  .DA %0.0.0.0.0.0.0.0.0.0.1.0.0.01 -- RELATIVE     26
0B03- 22 00             5720  .DA %0.0.0.0.0.0.0.0.0.1.0.0.0.10 -- LONG RELA.   28
                        5730 *-------------------------------
0B05- A4 D9 AC
0B08- A9 D3 D8
0B0B- AC A4 A8
0B0E- BE A3             5740 FMTSTR .AS -/$Y,)SX,$(>#/
                        5750 *-------------------------------
0B10- 20 8E FD          5760 INSDS1 JSR CROUT
0B13- A5 3B             5770        LDA PCH
0B15- 20 DA FD          5780        JSR PRBYTE
0B18- A5 3A             5790        LDA PCL
0B1A- 20 DA FD          5800        JSR PRBYTE
0B1D- A9 AD             5810        LDA #"-"
0B1F- 20 ED FD          5820        JSR COUT
0B22- A9 A0             5830        LDA #" "
0B24- 20 ED FD          5840        JSR COUT
0B27- A0 00             5850        LDY #0
0B29- B1 3A             5860        LDA (PCL),Y  GET OPCODE
0B2B- A8                5870 INSDS2 TAY          SAVE IN Y-REG
0B2C- B9 DB 08          5880        LDA OPINDEX,Y
0B2F- 0A                5890        ASL
0B30- AA                5900        TAX
0B31- 90 0F             5910        BCC .1          ...NOT SINGLE BYTE OPCODE
0B33- BD 17 08          5920        LDA OPNAMES.A,X
0B36- 85 2D             5930        STA RMNEM
0B38- BD 18 08          5940        LDA OPNAMES.A+1,X
0B3B- 85 2C             5950        STA LMNEM
0B3D- A9 00             5960        LDA #0
0B3F- 85 2F             5970        STA LENGTH
0B41- 60                5980        RTS
                        5990 *-------------------------------
0B42- BD 81 08          6000 .1     LDA OPNAMES.B,X
0B45- 85 2D             6010        STA RMNEM
0B47- BD 82 08          6020        LDA OPNAMES.B+1,X
0B4A- 85 2C             6030        STA LMNEM
0B4C- BE DB 09          6040        LDX OPFORMAT,Y
0B4F- BD DC 0A          6050        LDA FMTBL+1,X
0B52- 85 30             6060        STA FORMATH
0B54- BD DB 0A          6070        LDA FMTBL,X
0B57- 85 2E             6080        STA FORMATL
0B59- 29 03             6090        AND #3
0B5B- 85 2F             6100        STA LENGTH
0B5D- 8A                6110        TXA             CHECK IF IMMEDIATE
0B5E- D0 06             6120        BNE .2          ...NO
0B60- 24 00             6130        BIT IMM.SIZE CHECK IF 16-BIT MODE
0B62- 10 02             6140        BPL .2          ...NO
0B64- E6 2F             6150        INC LENGTH      ...YES
0B66- 60                6160 .2     RTS
```

```
                        6170 *--------------------------------
                        6180 INSTDSP
0B67- 20 10 0B          6190         JSR INSDS1
0B6A- A0 00             6200         LDY #0
0B6C- B1 3A             6210 .1      LDA (PCL),Y
0B6E- 20 DA FD          6220         JSR PRBYTE
0B71- A2 01             6230         LDX #1          PRINT 1 BLANK
0B73- 20 4A F9          6240 .2      JSR PRBL2
0B76- C4 2F             6250         CPY LENGTH
0B78- C8                6260         INY
0B79- 90 F1             6270         BCC .1
0B7B- A2 03             6280         LDX #3
0B7D- C0 04             6290         CPY #4
0B7F- 90 F2             6300         BCC .2
                        6310 *---PRINT MNEMONIC---------------
0B81- A0 03             6320         LDY #3
0B83- A9 06             6330 .3      LDA #6
0B85- 06 2D             6340 .4      ASL RMNEM
0B87- 26 2C             6350         ROL LMNEM
0B89- 2A                6360         ROL
0B8A- 10 F9             6370         BPL .4
0B8C- 20 ED FD          6380         JSR COUT
0B8F- 88                6390         DEY
0B90- D0 F1             6400         BNE .3
0B92- A4 2F             6410         LDY LENGTH
0B94- F0 33             6420         BEQ .8          ...SINGLE BYTE OPCODE
0B96- A5 2E             6430         LDA FORMATL
0B98- 29 20             6440         AND #$20        SEE IF SPECIAL
0B9A- D0 2E             6450         BNE .9          ...YES, MOVES OR RELATIVES
                        6460 *---PRINT NORMAL OPERANDS--------
0B9C- A9 A0             6470         LDA #" "
0B9E- 20 ED FD          6480         JSR COUT
0BA1- A2 0A             6490         LDX #10         11 FORMAT BITS
0BA3- 06 2E             6500 .5      ASL FORMATL
0BA5- 26 30             6510         ROL FORMATH
0BA7- 90 1D             6520         BCC .7
0BA9- BD 05 0B          6530         LDA FMTSTR,X
0BAC- 20 ED FD          6540         JSR COUT
0BAF- C9 A3             6550         CMP #"#"
0BB1- D0 07             6560         BNE .55
0BB3- 24 00             6570         BIT IMM.SIZE
0BB5- 10 0F             6580         BPL .7
0BB7- 20 ED FD          6590         JSR COUT
0BBA- C9 A4             6600 .55     CMP #"$"
0BBC- D0 08             6610         BNE .7
0BBE- B1 3A             6620 .6      LDA (PCL),Y
0BC0- 20 DA FD          6630         JSR PRBYTE
0BC3- 88                6640         DEY
0BC4- D0 F8             6650         BNE .6
0BC6- CA                6660 .7      DEX
0BC7- 10 DA             6670         BPL .5
0BC9- 60                6680 .8      RTS
                        6690 *---SPECIAL CASES----------------
0BCA- A9 A0             6700 .9      LDA #" "
0BCC- 20 ED FD          6710         JSR COUT
0BCF- A9 A4             6720         LDA #"$"
0BD1- 20 ED FD          6730         JSR COUT
0BD4- A5 2E             6740         LDA FORMATL
0BD6- 30 20             6750         BMI .11         MVN & MVP
0BD8- 88                6760         DEY             DISTINGUISH RELATIVES
0BD9- D0 07             6770         BNE .10         16-BIT RELATIVE
                        6780 *---8-BIT RELATIVE---------------
0BDB- C8                6790         INY             8-BIT RELATIVE
0BDC- B1 3A             6800         LDA (PCL),Y     GET 8-BIT OFFSET
0BDE- 38                6810         SEC
0BDF- 4C 38 F9          6820         JMP RELADR
                        6830 *---16-BIT RELATIVE--------------
0BE2- B1 3A             6840 .10     LDA (PCL),Y     LOW BYTE OF OFFSET
0BE4- 85 2E             6850         STA FORMATL
0BE6- C8                6860         INY
0BE7- B1 3A             6870         LDA (PCL),Y     HIGH BYTE OF OFFSET
0BE9- 85 30             6880         STA FORMATH
0BEB- 20 53 F9          6890         JSR PCADJ
0BEE- 18                6900         CLC
0BEF- 65 2E             6910         ADC FORMATL
0BF1- AA                6920         TAX
0BF2- 98                6930         TYA
0BF3- 65 30             6940         ADC FORMATH
0BF5- 4C 41 F9          6950         JMP PRNTAX
```

```
                6960 *---MVN & MVP--------------------
0BF8- B1 3A     6970 .11    LDA (PCL),Y
0BFA- 20 DA FD  6980        JSR PRBYTE
0BFD- A9 AC     6990        LDA #","
0BFF- 20 ED FD  7000        JSR COUT
0C02- A9 A4     7010        LDA #"$"
0C04- 20 ED FD  7020        JSR COUT
0C07- 88        7030        DEY
0C08- B1 3A     7040        LDA (PCL),Y
0C0A- 4C DA FD  7050        JMP PRBYTE
```

Finding Memory Size in ProDOS.....Bob Sander-Cederlof

On page 6-63 of Beneath Apple ProDOS there is a small piece of code designed to determine how much memory there is:

```
LDA $BF98
ASL
ASL
BIT 0
BPL SMLMEM      48K
BVS MEM128      128K
...             otherwise 64K
```

The code will not work. The BIT 0 will test bits 7 and 6 of memory location $0000, which have nothing whatsoever to do with how much memory is in your machine. What was intended was to test bits 7 and 6 of the A-register, or in other words bits 5 and 4 of $BF98. Here is one way you can do that:

```
LDA $BF98
ASL
ASL
ASL
BCC SMLMEM      48K
BMI MEM128      128K
...             OTHERWISE 64K
```

Notice that not only does this perform the test correctly, it is also one byte shorter!

If you insist on using the same number of bytes, here is another way to test those bits:

```
LDA $BF98
AND #%00110000  ISOLATE BITS 5 AND 4
CMP #%00100000
BCC SMLMEM      48K
BNE MEM128      128K
...             OTHERWISE 64K
```

If any of you have discovered any other problems with the sample code in this book, pass them along.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries. Back issues are available for $1.80 each (other countries add $1 per back issue for postage).